Video Compression Using EZW and FSBM

Sangeeta Mishra[*], **Sudhir Savarkar**[**]

[*] Research Scholar, Amravati
[**] DMCE, Airoli

Abstract- In this paper video compression is done using EZW as intra compression and seven different algorithms of the block matching algorithms used for motion estimation in video compression. It implements and compares 7 different types of block matching algorithms that range from the very basic Exhaustive Search to fast adaptive algorithms like Adaptive Rood Pattern Search. The algorithms that are evaluated in this paper are widely accepted by the video compressing community and have been used in implementing various standards, ranging from MPEG1 / H.261 to MPEG4 / H.263. The paper also presents a very brief introduction to the entire flow of video compression.

Index Terms- Block matching, motion estimation, video compression, MPEG, H.261, H.263,H.264, EZW.

I. Introduction

WITH the advent of the multimedia age and the spread of Internet, video storage on CD/DVD and streaming video has been gaining a lot of popularity. The ISO Moving Picture Experts Group (MPEG) video coding standards pertain towards compressed video storage on physical media like CD/DVD, where as the International Telecommunications Union (ITU) addresses real-time point-to-point or multi-point communications over a network. The former has the advantage of having higher bandwidth for data transmission. In either standard the basic flow of the entire compression decompression process is largely the same and is depicted in Fig. 1 shows the block diagram for video compression process. The most computationally expensive part in the compression process is the Motion Estimation[9]. Motion Estimation examines the movement of objects in sequence to try to obtain the vectors representing the estimated motion. Encoder side estimates the motion of the current frame with respect to previous frame. A motion compensated image of the current frame is then created. Motion vector is then transmitted to decoder. Decoder reverses the whole process and creates a full frame. This way motion compensation uses the knowledge of object motion to achieve data compression.
The encoding side estimates the motion in the current frame with respect to a previous frame. A motion compensated image for the current frame is then created that is built of blocks of image from the previous frame.

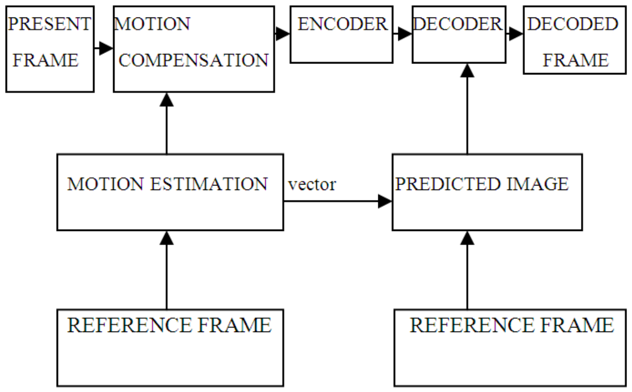

Fig. 1: Block Diagram for Video Compression process flow.

The motion vectors for blocks used for motion estimation are transmitted, as well as the difference of the compensated image with the current frame is also EZW encoded and sent. The encoded image that is sent is then decoded at the encoder and used as a reference frame for the subsequent frames. The decoder reverses the process and creates a full frame. The whole idea behind motion estimation based video compression is to save on bits by sending EZW encoded difference images which inherently have less energy and can be highly compressed as compared to sending a full frame that is EZW encoded.

This paper implements and evaluates the fundamental block matching algorithms along with EZW. The algorithms that have been implemented are Exhaustive Search (ES), Three Step Search (TSS), New Three Step Search (NTSS), Simple and Efficient TSS (SES), Four Step Search (4SS), Diamond Search (DS), and Adaptive Rood Pattern Search (ARPS). About EZW (Embedded Zerotrees of Wavelet Transforms) ,it is a lossy image compression algorithm. At low bit rates (i.e. high compression ratios) most of the coefficients produced by a subband transform (such as the wavelet transform) will be zero, or very close to zero. This occurs because "real world" images tend to contain mostly low frequency information (highly correlated). However where high frequency information does occur (such as edges in the image) this is particularly important in terms of human perception of the image quality, and thus must be represented accurately in any high quality coding scheme.

II. EZW

The embedded zerotree wavelet algorithm (EZW) is a simple, yet remarkably effective, image compression algorithm, having the property that the bits in the bit stream are generated in order of importance, yielding a fully embedded code. The embedded code represents a sequence of binary decisions that distinguish an image from the "null" image[10]. Using an embedded coding algorithm, an encoder can terminate the encoding at any point thereby allowing a target rate or target distortion metric to be met exactly. Also, given a bit stream, the decoder can cease decoding at any point in the bit stream and still produce exactly the same image that would have been encoded at the bit rate corresponding to the truncated bit stream. In addition to producing a fully embedded bit stream, EZW consistently produces compression results that are competitive with virtually all known compression algorithms on standard test images. Yet this performance is achieved with a technique that requires absolutely no training, no pre-stored tables or codebooks, and requires no prior knowledge of the image source.

III. THE EZW ALGORITHM

The EZW coding algorithm is based on following concepts:

i) A discrete wavelet transform or hierarchical subband decomposition.
ii) Prediction of the absence of significant information across scales by exploiting the self-similarity inherent in images.
iii) Entropy-coded successive-approximation quantization.
iv) Universal lossless data compression, which is achieved via adaptive arithmetic coding.

An EZW encoder was specially designed by J. M. Shapiro to use with wavelet transform[11]. EZW coding is more like a quantization method. It was originally designed to operate on images but it can also be used on other dimensional signals. The EZW encoder is based on progressive encoding to compress an image into a bitstream with increasing accuracy. This means that when more bits are added to the stream, the decoded image contain more detail, a property similar to JPEG encoded image. Progressive encoding is also called as embedded encoding which explains the 'E' letter in EZW. Coding an image using the EZW method together with some optimizations, results in a remarkably effective image compressor with the property that the compressed data stream can have any bit rate desired. Any bit rate is only possible if there is information loss somewhere so that the compressor is lossy. However lossless compression is also possible with EZW encoder with less spectacular results.

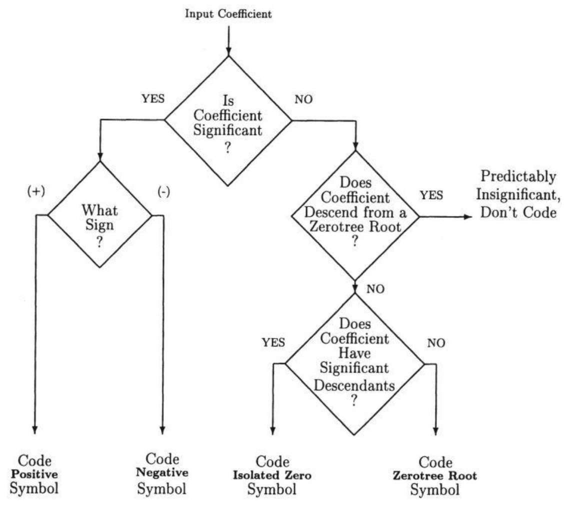
Input Coefficient
Is Coefficient Significant ?
YES
NO
What Sign ?
(+)
(-)
Does Coefficient Descend from a Zerotree Root Symbol ?
YES
Predictably Insignificant, Don't Code
NO
Does Coefficient Have Significant Descendants ?
YES
NO
Code Positive Symbol
Code Negative Symbol
Code Isolated Zero Symbol
Code Zerotree Root Symbol

Fig. 2 EZW flowchart to encode of DWT coefficients

In this method only four symbols are used to code the DWT coefficients, namely P, N, T and Z. The coding rules are as follows:

i) If the coefficient is larger than the threshold, P (positive significant) is coded.

ii) Else if the coefficient is smaller than threshold, N (negative significant) is coded.

iii) Else if the coefficient is the root of a zerotree, T (zerotree) is coded.

IV. BLOCK MATCHING ALGORITHMS

In BMA it is assumed that every pixel within a macro block has same motion activity and produce one motion vector for each macro block. The main idea behind block matching is to divide the current frame into number of macro blocks of fixed size and create a motion vector which comprises the location of the macro block of the current frame in the previous frame. Usually the macro block is taken as a sequence of 16 pixels and search area is up to 7 pixels on all fours sides of the corresponding macro block in previous frame[12]. The matching of one macro block with another is based on the output of a cost function. The macro block that results in the least cost is the one that matches the closest to current block. There are various cost functions, of which the most popular and less computationally expensive is Mean Absolute Difference (MAD) [9] given by equation (1).Another cost function is Mean Squared Error (MSE) [9] given by equation (2).

The underlying supposition behind motion estimation is that the patterns corresponding to objects and background in a frame of video sequence move within the frame to form corresponding objects on the subsequent frame. The idea behind block matching is to divide the current frame into a matrix of 'macro blocks' that are then compared with corresponding block and its adjacent neighbors in the previous frame to create a vector that stipulates the movement of a macro block from one location to another in the previous frame. This movement calculated for all the macro blocks comprising a frame, constitutes the motion estimated in the current frame. The search area for a good macro block match is constrained up to *p* pixels on all fours sides of the corresponding macro block in previous frame. This '*p*' is called as the search parameter. Larger motions require a larger *p*, and the larger the search parameter the more computationally expensive the process of motion estimation becomes. Usually the macro block is taken as a square of side 16 pixels, and the search parameter *p* is 7 pixels. The idea is represented in Fig 3. The matching of one macro block with another is based on the output of a cost function. The macro block that results in the least cost is the one that matches the closest to current block. There are various cost functions, of which the most popular and less computationally expensive is Mean Absolute Difference (MAD) given by equation (i). Another cost function is Mean Squared Error (MSE) given by equation (ii).

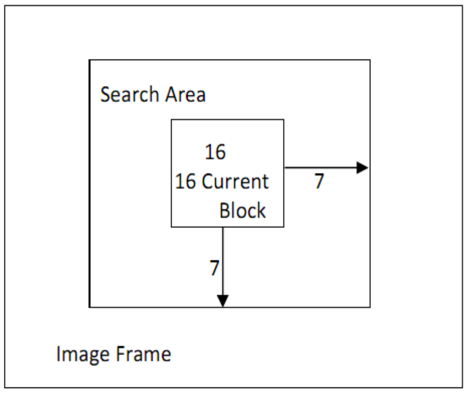

Fig 3: Block Matching of a 16x16 macro Block within a search area of 7 pixels

$$MAD = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left|C_{ij} - R_{ij}\right| \qquad \text{(i)}$$

$$MSE = \frac{1}{N^2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\left(C_{ij} - R_{ij}\right)^2 \qquad \text{(ii)}$$

where N is the side of the macro bock, Cij and Rij are the pixels being compared in current macro block and reference macro block, respectively.

Peak-Signal-to-Noise-Ratio (PSNR) given by equation (iii) characterizes the motion compensated image that is created by using motion vectors and macro clocks from the reference frame.

$$PSNR = 10Log_{10}\left[\frac{(\text{Peak to peak value of original data})^2}{MSE}\right] \qquad \text{(iii)}$$

A. Exhaustive Search (ES)

This algorithm, also known as Full Search, is the most computationally expensive block matching algorithm of all.

This algorithm calculates the cost function at each possible location in the search window. As a result of which it finds the best possible match and gives the highest PSNR amongst any block matching algorithm. Fast block matching algorithms try to achieve the same PSNR doing as little computation as possible. The disadvantage to ES is that the larger the search window gets the more computations it requires.

B. Three Step Search (TSS)

This is one of the earliest attempts at fast block matching algorithms and dates back to mid 1980s. It starts with the search location at the center and sets the 'step size' S = 4, for a usual search parameter value of 7. It then searches at eight locations +/- S pixels around location (0,0). From these nine locations searched so far it picks the one giving least cost and makes it the new search origin. It then sets the new step size S = S/2, and repeats similar search for two more iterations until S = 1.

C. New Three Step Search (NTSS)

NTSS [4] improves on TSS results by providing a center biased searching scheme and having provisions for half way stop to reduce computational cost. It was one of the first widely accepted fast algorithms and frequently used for implementing earlier standards like MPEG 1 and H.261. The TSS uses a uniformly allocated checking pattern for motion detection and is prone to missing small motions. In the first step 16 points are checked in addition to the search origin for lowest weight using a cost function. Of these additional search locations, 8 are a distance of S = 4 away (similar to TSS) and the other 8 are at S = 1 away from the search origin. If the lowest cost is at the origin then the search is stopped right here and the motion vector is set as (0, 0). If the lowest weight is at any one of the 8 locations at S = 1, then we change the origin of the search to that point and check for weights adjacent to it.

D. Simple and Efficient Search (SES)

SES [5] is another extension to TSS and exploits the assumption of unimodal error surface. The main idea behind the algorithm is that for a unimodal surface there cannot be two minimums in opposite directions and hence the 8 point fixed pattern search of TSS can be changed to incorporate this and save on computations. The algorithm still has three steps like TSS, but the innovation is that each step has further two phases.

E. Four Step Search (4SS)

Similar to NTSS, 4SS [6] also employs center biased searching and has a halfway stop provision. 4SS sets a fixed pattern size of S = 2 for the first step, no matter what the search parameter *p* value is. Thus it looks at 9 locations in a 5x5 window. If the least weight is found at the center of search window the search jumps to fourth step. If the least weight is at one of the eight locations except the center, then we make it the search origin and move to the second step. The search window is still maintained as 5x5 pixels wide. Depending on where the least weight location was, we might end up checking weights at 3 locations or 5 locations. Once again if the least weight location is at the center of the 5x5 search window we jump to fourth step or else we move on to third step. The third is exactly the same as the second step. In the fourth step the window size is dropped to 3x3, i.e. S = 1. The location with the least weight is the best matching macro block and the motion vector is set to point o that location.

F. Diamond Search (DS)

DS [7] algorithm is exactly the same as 4SS, but the search point pattern is changed from a square to a diamond, and there is no limit on the number of steps that the algorithm can take. DS uses two different types of fixed patterns, one is Large Diamond Search Pattern (LDSP) and the other is Small Diamond Search Pattern (SDSP). Just like in FSS, the first step uses LDSP and if the least weight is at the center location we jump to fourth step. The last step uses SDSP around the new search origin and the location with the least weight is the best match. As the search pattern is neither too small nor too big and the fact that there is no limit to the number of steps, this algorithm can find global minimum very accurately. The end result should see a PSNR close to that of ES while computational expense should be significantly less.

G. Adaptive Rood Pattern Search (ARPS)

ARPS [8] algorithm makes use of the fact that the general motion in a frame is usually coherent, i.e. if the macro blocks around the current macro block moved in a particular direction then there is a high probability that the current macro block will also have a similar motion vector. This algorithm uses the motion vector of the macro block to its immediate left to predict its own motion vector. The predicted motion vector points to (3, -2). In addition to checking the location pointed by the predicted motion vector, it also checks at a rood pattern distributed points, where they are at a step size of S = Max (|X|, |Y|). X and Y are the x-coordinate and y-coordinate of the predicted motion vector.

V. SIMULATION RESULTS

During the course of this project all of the above 7 algorithms have been implemented. ‘resi_gd’ video sequence with a distance of 2 between current frame and reference frame was used to generate the frame-by-frame results of the algorithms. Fig. 4 shows one of the original frame. Fig. 5 shows the compressed image using EZW. Fig.6 shows the Reconstructed image using artificial colors. The original size of the video is 25MB and the reconstructed video is of size 21.3MB. The PSNR comparison of the compensated images generated using the algorithms is shown in Fig 7.

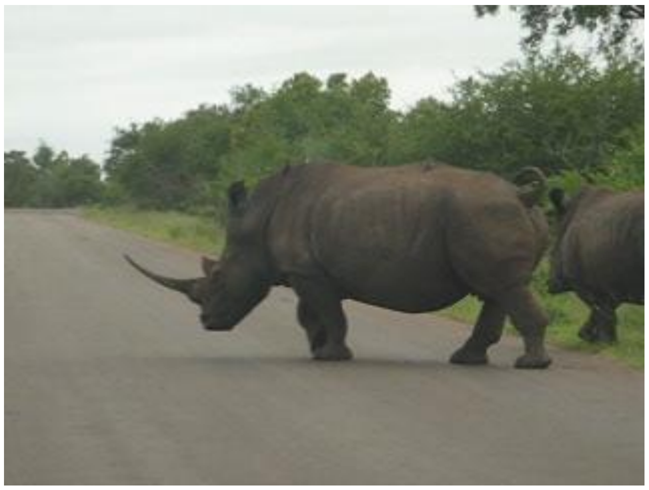

Fig. 4 Original Frame

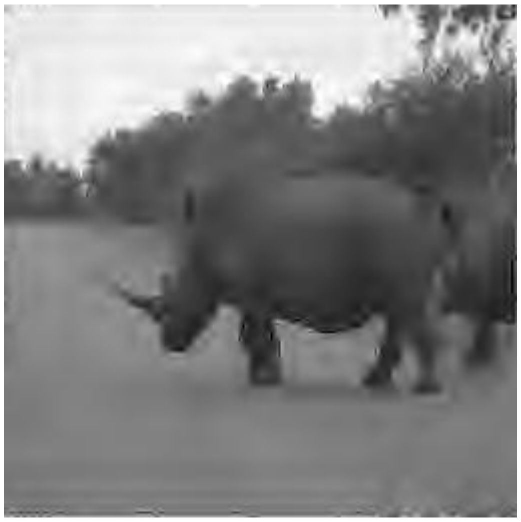

Fig. 5 Compressed Frame using EZW

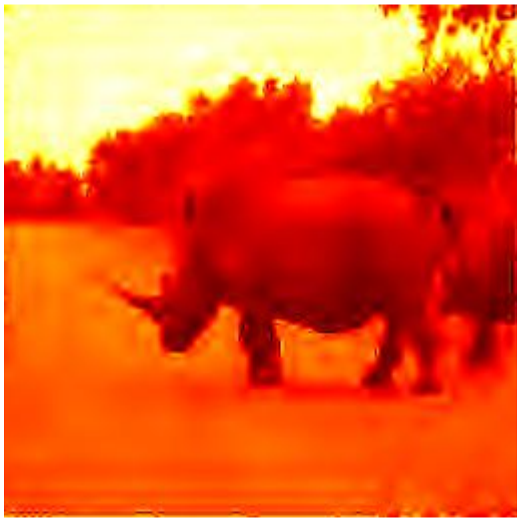

Fig.6 Reconstructed Frame (using artificial color)

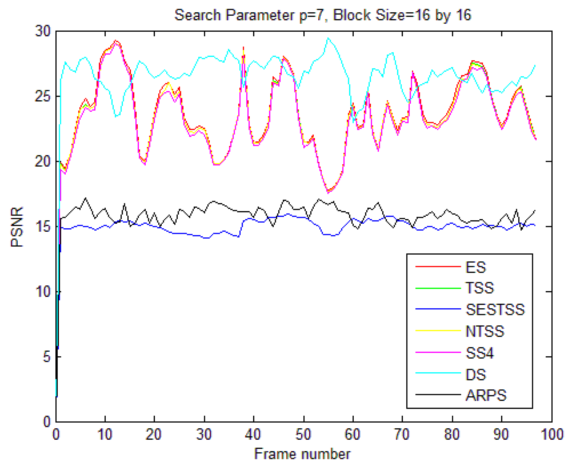

Fig.7 Plot of PSNR and Frame Number

As is shown by Fig. 7, SS4, ES, NTSS and TSS come pretty close to the PSNR results of ES. While the ES takes on an average around ~200 searches per macro block, DS and 4SS drop that number by more than an order of magnitude. ARPS further drops by a factor of 2 compared to DS. NTSS and TSS although do not come close in PSNR performance to the results of ES, but even they drop down the number of computations required per macro block by almost an order of magnitude. SES takes up less number of search point computations amongst all but ARPS. The computations for different algorithms are mentioned

TSS→23.046875,ES→199.515625,SS4→24.359375,
ARPS→17.15234375, DS→27.90625, NTSS→30.2265625
SESTSS→15.91796875

VI. V. CONLUSION & FUTURE WORK

As from the results it is clear that the clarity of the image is poor and if the color used were from the original image then quality can be improved. The results are much better if instead of EZW,SPIHT algorithm is used. Before compression the size of the clip was 25MB and after compression the size is reduced to 21.3MB. So the compression is achieved with a good compression ratio and compression factor.

REFERENCES

[1] Borko Furht, Joshua Greenberg, Raymond Westwater, Motion Estimation Algorithms For Video Compression. Massachusetts: Kluwer Academic Publishers, 1997. Ch. 2 & 3.

[2] M. Ghanbari, *Video Coding, An Introduction to Standard Codecs*, London: The Institute of Electrical Engineers, 1999. Ch.2, 5, 6, 7 & 8

[3] [3] Iain E. G. Richardson, Video Codec Design, West Sussex: John Wiley & Sons Ltd., 2002, Ch. 4, 5, & 6.

[4] Renxiang Li, Bing Zeng, and Ming L. Liou, "A New Three-Step Search Algorithm for Block Motion Estimation", *IEEE Trans. Circuits And Systems For Video Technology*, vol 4., no. 4, pp. 438-442, August 1994.

[5] Jianhua Lu, and Ming L. Liou, "A Simple and Efficent Search Algorithm for Block-Matching Motion Estimation", *IEEE Trans.* Circuits And Systems For Video Technology, vol 7, no. 2, pp. 429-433, April 1997

[6] Lai-Man Po, and Wing-Chung Ma, "A Novel Four-Step Search Algorithm for Fast Block Motion Estimation", *IEEE Trans. Circuits And Systems For Video Technology*, vol 6, no. 3, pp. 313-317, June 1996.

[7] Shan Zhu, and Kai-Kuang Ma, " A New Diamond Search Algorithm for Fast Block-Matching Motion Estimation", *IEEE Trans. Image Processing*, vol 9, no. 2, pp. 287-290, February 2000.

[8] Yao Nie, and Kai-Kuang Ma, "Adaptive Rood Pattern Search for Fast Block-Matching Motion Estimation", *IEEE Trans. Image Processing*, vol 11, no. 12, pp. 1442-1448, December 2002.

[9] Aroh Barjatya, " Block Matching Algorithms For Motion Estimation" DIP 6620 Spring 2004.

[10] http://en.wikipedia.org/wiki/Embedded_Zero_Tree.

[11] J. M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Trans. on Signal Processing*, Vol. 41, No. 12, pp. 3445– 3462, Dec. 1993.

AUTHORS

First Author – Sangeeta Mishra, Research Scholar, Amravati 09867798525, sangeeta.mishra@thakureducation.org
Second Author – Sudhir Savarkar, DMCE, Airoli, 09819768930, sudhir.savarkar@yahoo.com